(19) Deutsches Patent- und Markenamt

(10) **DE 10 2012 102 254 B4** 2020.09.24

(12) # Patentschrift

(21) Aktenzeichen: **10 2012 102 254.2**
(22) Anmeldetag: **16.03.2012**
(43) Offenlegungstag: **19.09.2013**
(45) Veröffentlichungstag der Patenterteilung: **24.09.2020**

(51) Int Cl.: ***H04L 9/32*** (2006.01)

Innerhalb von neun Monaten nach Veröffentlichung der Patenterteilung kann nach § 59 Patentgesetz gegen das Patent Einspruch erhoben werden. Der Einspruch ist schriftlich zu erklären und zu begründen. Innerhalb der Einspruchsfrist ist eine Einspruchsgebühr in Höhe von 200 Euro zu entrichten (§ 6 Patentkostengesetz in Verbindung mit der Anlage zu § 2 Abs. 1 Patentkostengesetz).

(73) Patentinhaber:
Infineon Technologies AG, 85579 Neubiberg, DE

(74) Vertreter:
Zimmermann & Partner Patentanwälte mbB, 80331 München, DE

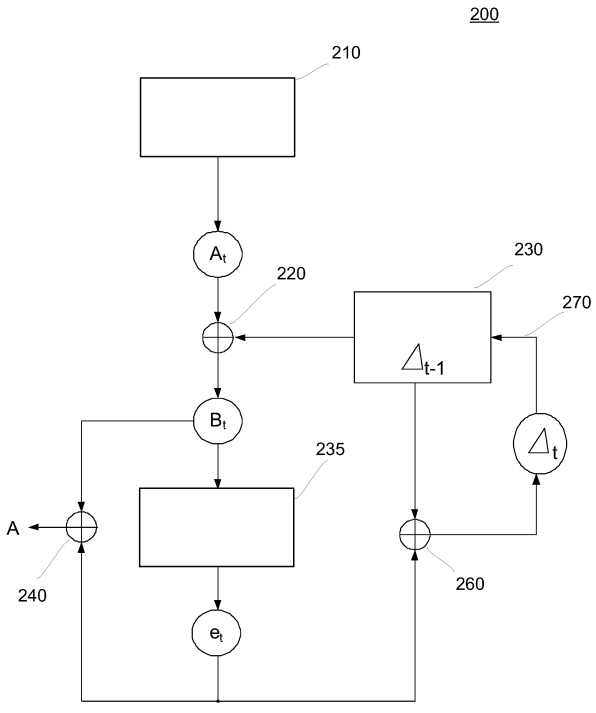
(72) Erfinder:
Göttfert, Rainer, Dr., 85640 Putzbrunn, DE;
Gammel, Berndt, Dr., 85570 Markt Schwaben, DE;
Künemund, Thomas, Dr., 80809 München, DE;
Dirscherl, Gerd, 81541 München, DE

(56) Ermittelter Stand der Technik:

US	**2011 / 0 002 461**	**A1**
WO	**2006/ 129 242**	**A2**
WO	**2010/ 100 015**	**A1**

(54) Bezeichnung: **Vorrichtung und Verfahren zur Rekonstruktion einer Bitfolge unter Vorkorrektur**

(57) Hauptanspruch: Verfahren (100, 200) zur Rekonstruktion einer PUF A für die Nutzung in einem elektronischen Gerät, umfassend:
- Erzeugen (110) einer potenziell fehlerbehafteten PUF A_t;
- Vorkorrigieren (120, 220) der PUF A_t mittels eines gespeicherten Korrekturvektors $Delta_{t-1}$, um eine vorkorrigierte PUF B_t zu erhalten;
- Rekonstruieren (130, 235) der PUF A aus der vorkorrigierten PUF B_t mittels eines Fehlerkorrekturalgorithmus, umfassend:
- Anwenden (235) des Fehlerkorrekturalgorithmus auf B_t, um einen Fehlervektor e_t zu erhalten;
- XOR-Verknüpfen (240) von B_t und e_t,
wobei ein neuer Korrekturvektor Deltat durch XOR-Verknüpfung (260) des gespeicherten Korrekturvektors $Delta_{t-1}$ und des Fehlervektors e_t berechnet wird, und wobei Deltat als neuer Korrekturvektor in einem Fehlerregister abgespeichert (270) wird, der bei einer späteren erneuten PUF-Rekonstruktion als Fehlervektor $Delta_{t-1}$ dient.



Beschreibung

[0001] Die Erfindung liegt auf dem Gebiet der Kryptographie, und Aspekte der Erfindung betreffen eine Vorrichtung und ein Verfahren zur Rekonstruktion einer Physically Uncloneable Function (PUF), insbesondere zur Nutzung in einer elektronischen Chipkarte oder einem RFID-Gerät.

[0002] Die US 2011 / 0 002 461 A1 beschreibt ein System zum Sichern eines integrierten Schaltkreischips, der für biometrische Sensoren oder andere elektronische Geräte verwendet wird, indem ein Schaltkreis mit physikalisch nicht klonbarer Funktion (PUF) verwendet wird. Diese PUF-Funktionen werden wiederum zur Erzeugung von Sicherheitswörtern und -schlüsseln verwendet, wie z.B. ein öffentlicher oder privater RSA-Schlüssel. Ein solches System kann dazu verwendet werden, biometrische Sicherheitssensoren und IC-Chips, wie Fingerabdrucksensoren und Sensortreiberchips, vor Angriffen oder Spoofing zu schützen. Das System kann auch in einer effizienten Methode verwendet werden, um einzigartige Sicherheitsschlüssel für die Geräteeinstellung oder die Authentisierung beim Einschalten zu erzeugen. Diese Schlüssel können auf einer Niedrigfrequenzbasis erzeugt und dann häufig für spätere Sicherheitsüberprüfungen wiederverwendet werden. Im Betrieb können die gespeicherten Schlüssel verwendet werden, um das Gerät effizient zu authentifizieren, ohne dass jedes Mal aufwändige Prozesse zur Generierung von Sicherheitsschlüsseln durchgeführt werden müssen, wobei gleichzeitig eine gute Gerätesicherheit erhalten bleibt.

[0003] Weiterhin beschreibt die WO 2006 / 129 242 A2 eine Methode zur Authentifizierung eines physischen Objekts mit Hilfe von ersten Hilfsdaten und einem ersten Kontrollwert, der mit einem Referenzobjekt verbunden ist. Das Verfahren umfasst einen Schritt zum Erzeugen eines ersten Eigenschaftssatzes unter Verwendung einer rauschkompensierenden Abbildung über Informationen, die aus den ersten Hilfsdaten und metrischen Daten, die dem physikalischen Objekt zugeordnet sind, abgeleitet werden, und einen Schritt zum Herstellen einer ausreichenden Übereinstimmung zwischen dem physikalischen Objekt und dem Referenzobjekt unter Verwendung des ersten Eigenschaftssatzes und des ersten Kontrollwertes. Das Verfahren umfasst ferner einen Schritt zum Erzeugen von Aktualisierungsdaten zum Aktualisieren der ersten Hilfsdaten unter Verwendung der ersten Hilfsdaten, des ersten Eigenschaftssatzes und der metrischen Daten. Außerdem wird ein Client-Terminal bereitgestellt, das zum Ausführen der Methode konfiguriert ist.

[0004] Die Abkürzung PUF steht für Physically Uncloneable Function, auch als physikalische Hashfunktion bezeichnet. Das zugrundeliegende Konzept ist, physikalische Eigenschaften eines Objektes zu digitalisieren und so eine dem Objekt zugeordnete Bitfolge zu erhalten. Dabei ist es wünschenswert, dass die Bitfolgen zweier verschiedener physikalischer Objekte zueinander unkorreliert sind. Ein einfaches Beispiel zur Veranschaulichung ist ein Blatt Papier. Unter einem Mikroskop betrachtet, erkennt man eine spezielle Feinstruktur der Holzspäne oder Zellstoffteile. Die Struktur wird vermessen und mit Hilfe eines geeigneten Algorithmus als Bitfolge dargestellt. Diese Bitfolge ist dann die dem Blatt Papier zugeordnete PUF. Ein anderes Blatt Papier wird im Allgemeinen eine gänzlich andere Bitfolge ergeben, das heißt eine Bitfolge, die unkorreliert ist zu der Bitfolge des ersten Blatts. Die Begriffe „Bitfolge“ und „Bitstring“ werden im Folgenden synonym verwendet.

[0005] Den Prozess der Erzeugung einer Bitfolge (der PUF) aus den Eigenschaften des physikalischen Objektes nennt man PUF-Generierung. Eine Hauptverwendung von PUFs ist die Herstellung kryptographischer Schlüssel für vollelektronische beziehungsweise computerisierte Verschlüsselungsverfahren. Z.B. könnte man den PUF-Bitstring selbst als kryptographischen Schlüssel verwenden. Oder man könnte, was bestimmte Vorteile hat, den PUF-Bitstring zu einem kürzeren Bitstring komprimieren und letzteren als kryptographischen Schlüssel verwenden. Letzteres Verfahren wird üblicherweise bei Chipkarten angewendet, wobei ein Mechanismus zur PUF-Generierung in der Elektronik der Karte integriert ist. Auf diese Weise wird über die PUF-Generierung und deren Nutzung zur Schlüsselerzeugung vermieden, dass der Schlüssel selbst auf der Karte abgespeichert werden muss, was ein Sicherheitsrisiko darstellen würde.

[0006] Eine wünschenswerte Eigenschaft eines PUF-Mechanismus ist, dass dasselbe physikalische Objekt, also etwa dieselbe Chipkarte, jedesmal dieselbe Bitfolge ergibt im Zuge einer erneuten PUF-Generierung. Dies soll insbesondere auch unter verschiedenen Umweltbedingungen zutreffen, wie Temperatur, Luftfeuchtigkeit, Helligkeit, elektrische und magnetische Feldstärken usw.

[0007] Das ist im Allgemeinen jedoch nicht der Fall. Eine wiederholte PUF-Generierung für dasselbe physikalische Objekt liefert im Allgemeinen unterschiedliche Bitfolgen. Die Bitfolgen sind zwar untereinander durchaus ähnlich, aber nicht unbedingt miteinander identisch. Dieses Defizit versucht man mit Methoden der Codierungstheorie (Fehlerkorrektur) auszugleichen.

[0008] Man geht dabei wie folgt vor. Gegeben ist ein physikalisches Objekt. Zu Beginn erzeugt man die dem Objekt zugeordnete PUF-Bitfolge A. Der Bitstring A ist also das Ergebnis der ersten PUF-Generierung. Die Bitfolge A wird wie eine Nachricht in der Codierungstheorie betrachtet, die über einen störanfälligen Kanal übertragen werden soll, wobei zu erwarten ist, dass bei der Übertragung Fehler auftreten, d.h. einzelne Biteinträge umfallen, das heißt eine Null wird zu einer Eins oder umgekehrt. In der Codierungstheorie begegnet man diesem Problem, indem man die Nachricht A mit einer Redundanz R versieht und das Codewort (A, R) überträgt. Wenn bei der Übertragung Fehler auftreten, so können diese dank der Redundanz R mit Methoden der Codierungstheorie korrigiert werden. Nach der Korrektur liegt das fehlerfreie Nachrichtenwort A wieder vor.

[0009] Dasselbe Konzept macht man sich in der PUF-Generierung zunutze. Der ursprüngliche PUF-Wert A (der Wert, der bei der ersten PUF-Generierung anfällt) wird als der wahre PUF-Wert bezeichnet. Aus dem wahren PUF-Wert A wird ein zugehöriger Redundanzwert R berechnet. R wird Hilfsinformation genannt, und mit der Hilfe von R soll - zu einem späteren Zeitpunkt - die Rekonstruktion des wahren PUF-Werts A gelingen.

[0010] Der Einfachheit halber wurde hier angenommen, dass der wahre PUF-Wert A derjenige Bitstring sei, der bei der allerersten PUF-Generierung anfällt. Tatsächlich wird etwa der wahre PUF-Wert einer Chipkarte in der Fertigung im Zuge der Chip-Personalisierung bestimmt. Dabei ist es üblich, mehrmals bzw. häufig hintereinander einen PUF-Wert zu erzeugen, und etwa den Mittelwert oder den häufigsten Wert als den wahren PUF-Wert zu definieren. Eine andere Vorgehensweise ist, eine Reserve einzuplanen. Angenommen, man benötigt einen 800 Bit langen PUF-Wert. Man erzeugt jedoch (beispielhaft) einen 1000 Bit langen PUF-Wert, um die Reserve zu haben. In der Fabrik wird dann mehrmals, zum Beispiel 100 Mal, der 1000 Bit PUF-Wert erzeugt. Jede Bit-Position, die während dieser 100 Erzeugungen nicht stabil ist, also nicht immer den gleichen Bitwert zeigt, wird als ungültig erklärt. Angenommen, es gibt 840 Stellen, an denen während der 100 PUF-Generierungen jedesmal derselbe Bitwert auftrat. Dann werden von diesen 840 Stellen zum Beispiel 800 ausgewählt, und diese 800 Stellen definieren den wahren PUF-Wert.

[0011] Der mit Hilfe des Codierungsalgorithmus berechnete Wert R wird abgespeichert. Aus Sicherheitsgründen wird der PUF-Wert A selbst nicht abgespeichert und steht daher auch nicht immer zur Verfügung. Grund ist, dass der PUF-Wert A direkt als kryptographischer Schlüssel verwendet wird, oder aus ihm ein kryptographischer Schlüssel abgeleitet wird. Wenn der PUF-Wert A leicht zugänglich wäre, könnte man den zugeordneten kryptographischen Schlüssel nicht mehr als geheim ansehen. Bei einer späteren erneuten PUF-Generierung erhält man einen neuen PUF-Wert B. Der Wert B ist im Allgemeinen nicht identisch mit A, unterscheidet sich aber von A nur geringfügig. Das Ziel ist, den wahren PUF-Wert A wiederzugewinnen aus dem verfügbaren Wert B.

[0012] Dies gelingt mit der Hilfe von R und Methoden der Codierungstheorie:

$$B->(B,R)->(A,R)->A$$

[0013] Der aktuelle und vorliegende PUF-Wert B wird also um die Hilfsinformation R erweitert, wobei A, B und R Bitstrings sind. Die Bitfolge (B, R) wird dann als fehlerhaftes Wort im Rahmen der Codierungstheorie betrachtet und dann mittels der Codierungstheorie der Fehler korrigiert. Man erhält das fehlerbereinigte Wort (A, R). Insbesondere verfügt man nun über den wahren PUF-Wert A.

[0014] Die Aufgabe, den wahren PUF-Wert A aus dem zuletzt erzeugten und gegenwärtig vorliegenden PUF-Wert B zu rekonstruieren, gelingt nur dann, wenn B sich nicht zu stark von A unterscheidet. In der Terminologie der Codierungstheorie: Wenn bei der Generierung von B nicht zu viele Fehler passiert sind, relativ zu dem ursprünglichen wahren PUF-Wert A betrachtet.

[0015] Es hängt von der technischen Realisierung einer PUF ab, wie sehr sich ein neu generierter PUF-Wert B von dem wahren PUF-Wert A typischerweise unterscheidet, wie viele Fehler also typischerweise zu korrigieren sind. Je nach technischer Realisierung der PUF wird sich B von A potenziell in weniger als 1 % der Positionen unterscheiden, etwa in 0,3 % oder 0,6 %, oder in bis zu 25 %. Je stärker sich B im Durchschnitt von A unterscheidet, um so größer und kostspieliger ist die Hardware-Realisierung des PUF-Rekonstruktionsalgorithmus. Dies bedeutet auch höhere Herstellungskosten, einen höheren Platzbedarf und eventuell einen höheren Energieverbrauch.

[0016] Dafür gibt es mehrere Gründe. Wenn etwa ein 128 Bit langer geheimer Schlüssel aus dem PUF-Wert gebildet werden soll, ergeben sich folgende Parameter:

[0017] Je höher die Fehlerrate ist (also je stärker sich B von A unterscheidet), umso länger müssen die Bitstrings A und B sein, um am Ende zu einem sicheren 128-Bit-Schlüssel zu führen. Wenn z.B. 15 % Fehler auftreten in B gegenüber A, dann muss A (und somit auch B) ca. 4000 Bit lang sein, um einen 128 Bit geheimen Schlüssel abzuwerfen (bei 25 % Fehlern würden etwa 6000 Bit benötigt). Wenn nur 1 % Fehler auftreten, müssten A und B circa 600 Bit lang sein, um ebenfalls einen 128 Bit langen geheimen Schlüssel zu liefern. Die Berechnung der oben angegebenen Werte und Verhältnisse wird mit Hilfe der Codierungstheorie ausgeführt und ist dem Fachmann bekannt, sie soll daher hier nicht näher ausgeführt werden. Bei noch geringerer Fehlerrate und einem kürzeren zu erzeugenden kryptographischen Schlüssel könnten z.B. auch A und B zu je 64 Bit ausreichen.

[0018] Je mehr Fehler passieren, umso stärker muss der verwendete FehlerkorrekturAlgorithmus sein, und umso aufwändiger und damit teurer ist seine Implementierung.

[0019] Im Bereich der elektronischen Chipkarten werden üblicherweise Verfahren eingesetzt, bei denen die PUF durch Messungen an in Silizium implementierten elektronischen Schaltkreisen erzeugt wird, zum Beispiel Transistoren. Unterschiede im Herstellungsprozess der Chips, die auch der Hersteller nicht völlig unter Kontrolle hat, sind dafür verantwortlich, dass zwei unterschiedliche Chips zu zueinander unkorrelierten von ihnen erzeugten PUF-Strings führen. Dies wird dafür genutzt, dass verschiedene Chips automatisch verschiedene PUFs generieren, was sicherheitstechnisch eine Grundvoraussetzung ist.

[0020] Für solche Schaltkreise ist typischerweise mit einer Fehlerrate zwischen 1 % und 10 % zu rechnen. Das heißt, der neu generierte PUF-String B wird sich von dem wahren PUF-String A potenziell in etwa p % der Bit-Stellen unterscheiden, wobei p eine Zahl zwischen 1 und 10 ist. Entsprechend dem durch Experimente bestimmten Wert p wird dann die passende PUF-String-Länge und ein erforderlicher Fehlerkorrekturalgorithmus implementiert.

[0021] In einem Beispiel soll ein 128 Bit langer kryptographischer Schlüssel aus dem PUF-String A extrahiert werden. Wenn p gleich 1 ist, dann muss hierfür der PUF-String A in etwa die Länge 600 Bit haben. Wenn p gleich 10 ist, muss der PUF-String A in etwa 3000 Bit lang sein. Darüber hinaus gilt, dass für p gleich 1 ein einfacherer Fehlerkorrekturalgorithmus erforderlich ist beziehungsweise genügt, als für p gleich 10.

[0022] Wie bereits oben beschrieben, wird dieses Problem derzeit allein mit Methoden der Codierungstheorie gelöst: Es wird ein passender algebraischer Code gewählt, der fast immer ein linearer Code ist. Zu dem ursprünglich gemessenen PUF-Bit-String A wird mit dem gewählten Code der zugehörige Redundanzwert R berechnet. Dieser Redundanzwert R wird dannstellvertretend für den PUF-String A - im NVM (nichtflüchtiger Speicher) der Chipkarte gespeichert. A selbst wird aus Sicherheitsgründen nicht gespeichert. Bei einer erneuten späteren PUF-Generierung erhält man den Bit-String B. Mithilfe des Redundanzwerts R und einem Algorithmus aus der Theorie der fehlerkorrigierenden Codes (Algebraische Codierungtheorie) wird dann aus B der Wert A berechnet. In anderen Worten: Man fasst B als eine fehlerhafte Version von A auf, und korrigiert die Fehler mit Hilfe von R und dem Fehlerkorrekturalgorithmus.

[0023] Dabei ist zu beachten, dass die Redundanz R kürzer sein muss als A. Da R im NVM abgespeichert ist, gilt R als mehr oder weniger öffentlich bekannt. Aus A wird ein kryptographischer Schlüssel extrahiert. Sei |A| die Bitlänge von A, und sei |R| die Bitlänge von R. Dann ist die Differenz |A| - |R| die Länge des Geheimnisses. Und es kann nur ein einziger kryptographischer Schlüssel dieser Länge aus A gewonnen werden.

[0024] In einem Beispiel habe der PUF-Wert A eine Länge von 500 Bit. Die Redundanz R habe 400 Bit. Dann kann aus A ein 100 Bit langer kryptographischer Schlüssel abgeleitet werden. Eine naheliegende Möglichkeit dies zu tun, besteht darin, dass man jeweils 5 Bit von A modulo 2 addiert - das heißt, XOR-verknüpft - um ein Schlüsselbit zu gewinnen.

[0025] Vor diesem Hintergrund besteht ein Bedarf nach Verfahren und Vorrichtungen, die eine verbesserte PUF-Generierung ermöglichen, etwa schneller oder mit vermindertem Rechenaufwand.

[0026] Vor diesem Hintergrund wird daher ein Verfahren zur Rekonstruktion einer PUF für die Nutzung in einer Chipkarte gemäß Anspruch 1, und eine Vorrichtung gemäß Anspruch 18 vorgeschlagen. Weitere bevorzugte Aspekte der Erfindung ergeben sich aus den abhängigen Ansprüchen, aus den Figuren und aus der Beschreibung.

[0027] Im Weiteren soll die Erfindung anhand von in Figuren dargestellten Ausführungsbeispielen erläutert werden, aus denen sich weitere Vorteile und Abwandlungen ergeben.

Fig. 1 zeigt schematisch ein Verfahren zur Rekonstruktion einer PUF gemäß Ausführungsformen der Erfindung;

Fig. 2 zeigt eine schematische Darstellung einer Vorrichtung gemäß Ausführungsformen der Erfindung;

Fig. 3 zeigt schematisch eine Vorrichtung gemäß Ausführungsformen.

[0028] Im Folgenden werden verschiedene Ausführungsformen der Erfindung beschrieben, von denen einige auch in den Figuren beispielhaft dargestellt sind. Bei der folgenden Beschreibung der Figuren beziehen sich gleiche Bezugszeichen auf gleiche oder ähnliche Komponenten. Im Allgemeinen werden nur Unterschiede zwischen verschiedenen Ausführungsformen beschrieben. Hierbei können Merkmale, die als Teil einer Ausführungsform beschrieben werden, auch ohne weiteres im Zusammenhang mit anderen Ausführungsformen kombiniert werden, um noch weitere Ausführungsformen zu erzeugen.

[0029] Wie bereits erwähnt, wird die Rekonstruktion des wahren PUF-Bit-Strings A aus dem vorliegenden Bit-String B mit Hilfe der Redundanz R mit einem Fehlerkorrekturalgorithmus bewerkstelligt. Es besteht ein enger Zusammenhang zwischen dem Problem der PUF-Generierung und der PUF-Rekonstruktion einerseits, und dem Problem der Übertragung einer Nachricht über einen störanfälligen Kanal und der Korrektur der empfangenen Nachricht andererseits.

[0030] Bei der PUF-Rekonstruktion hat man ursprünglich den (wahren) PUF-Bit-String A generiert. Zu einem späteren Zeitpunkt, im Zuge einer neuen PUF-Rekonstruktion, erhält man den Bit-String B. Aus dem vorliegenden B soll das (dann) unbekannte A rekonstruiert werden.

[0031] Bei der Nachrichtenübertragung hat man eine Nachricht A, die man immer auch als Bit-String darstellen kann. A wird über einen störanfälligen Kanal geschickt. Es treten möglicherweise einige Übertragungsfehler auf. Eine Nachricht B wird empfangen. Mit einem Fehlerkorrekturalgorithmus werden dann die aufgetretenen Fehler in B korrigiert. Als Folge erhält man die ursprüngliche Nachricht A. Bei der Nachrichtenübertragung wird der Kanal üblicherweise als sogenannter „Binary Symmetrie Channel“ angenommen. Das bedeutet, man nimmt an, dass die einzelnen Nachrichtenbits in A voneinander unabhängig mit derselben Wahrscheinlichkeit „umkippen“, das heißt zu einem fehlerhaften Bit werden. Für einen Binary Symmetric Channel mit dem Parameter p gilt also: Ein einzelnes beobachtetes, jedoch beliebiges Nachrichtenbit wird bei der Übertragung mit der Wahrscheinlichkeit p umgewandelt in sein Komplementärbit (aus 0 wird 1, beziehungsweise aus einer 1 wird eine 0). Mit der Wahrscheinlichkeit **1-p** wird das Nachrichtenbit fehlerfrei übertragen.

[0032] Betrachtet man nun die PUF-Situation, so stellt man abweichend von der obigen Betrachtung für die Nachrichtenübertragung fest, dass dabei nicht jedes Bit von A mit derselben Wahrscheinlichkeit im Zuge einer neuen PUF-Generierung in sein Komplementärbit umgewandelt wird. Vielmehr zeigen Beobachtungen, dass die einzelnen Bits in A mit unterschiedlichen Wahrscheinlichkeiten „umfallen“, das heißt in ihr jeweiliges Komplementärbit umgewandelt werden. In der Konsequenz werden Bits der Folge, die sehr häufig umfallen, daher von Anfang an - also beim Herstellungsprozess des Chips in der Fabrik - identifiziert, als ungültig/unbrauchbar erklärt und in Zukunft ignoriert. Aber auch unter den verbleibenden brauchbaren Bits stellt man fest, dass sie mit unterschiedlichen Wahrscheinlichkeiten umfallen. Es scheint sogar Bits zu geben, die nie umfallen. Dies sind stabile Bits, die immer den richtigen Wert annehmen. Darüber hinaus gibt es die folgenden zwei Phänomene:

[0033] Erstens verändern einzelne brauchbare Bits (also Bits, die ursprünglich nur sehr selten umfielen) ihr Verhalten als Folge eines Alterungsprozesses dahingehend, dass sie zunehmend häufiger umfallen. Im Extremfall fallen sie irgendwann dauernd um, das heißt sie sind stabil, liefern aber immer den falschen Wert.

[0034] Zweitens variiert die Wahrscheinlichkeit dafür, dass ein Bit umfällt, unter dem Einfluss von Umweltbedingungen. Vor allem gibt es bei Silizium-basierten, also mittels Transistor-basierter Schaltkreise erzeugter PUFs Temperaturabhängigkeiten. So gibt es Bits, die bei -20 Grad Celsius sehr selten umfallen, bei +80 Grad Celsius dafür aber sehr häufig umfallen. Auf diese Weise können Bits, die nahezu stabil sind unter einer Umweltbedingung, sehr instabil werden unter veränderten Umweltbedingungen, vor allem wenn zwischen der alten und der neuen Umweltbedingung starke Unterschiede bestehen.

[0035] Das bedeutet, dass die Annahme falsch ist, dass sich eine neue PUF-Generierung (relativ zum wahren PUF-Bit-String A) ebenso verhalten würde wie die Nachrichtenübertragung über einen Binary Symmetric Channel. Vielmehr muss man davon ausgehen, dass die PUF-Generierung eine Art Memory Effect aufweist.

Der Fehler, der bei der PUF-Generierung zum Zeitpunkt t auftritt, ist in den meisten Fällen ähnlich dem Fehler, der bei der PUF-Generierung zum Zeitpunkt t-1 auftrat.

[0036] Folglich sind aufgetretene Fehler bei PUF-Generierungen, die in nahe beieinander liegenden Zeitpunkten durchgeführt werden (unter gleichen oder zumindest ähnlichen Umweltbedingungen), nicht statistisch voneinander unabhängig - wie das im allgemeinen bei der oben beschriebenen Nachrichtenübertragung über einen Kanal der Fall ist - sondern sie sind vielmehr miteinander verwandt.

[0037] Dieser Effekt wird in dem hier beschriebenen Verfahren gemäß Ausführungsbeispielen genutzt: Der Fehler, der bei der PUF-Rekonstruktion zum früheren Zeitpunkt t-1 berechnet wurde, wird gespeichert, und für die PUF-Rekonstruktion zum späteren bzw. aktuellen Zeitpunkt t genutzt.

[0038] Dabei wird der Umstand ausgenutzt, dass die PUF-Generierung in einem bestimmten Ausmaß von Umweltbedingungen abhängt. Es ist daher im allgemeinen leichter, einen aufgetretenen PUF-Fehler relativ zu einem aus dem unmittelbar vorangegangenen PUF-Request generierten PUF-Wert zu korrigieren, als relativ zum allerersten (im Lebenszyklus des Chips oder zum Beispiel der Chipkarte) generierten PUF Wert.

[0039] Es sei A der wahre PUF-Wert, also der allererste PUF-Wert, der in der Fabrik bei der Herstellung des Chips bestimmt wurde.

[0040] Weiter sei A_t der anfallende PUF-Wert zum (späteren) Zeitpunkt t. Das heißt, zum Zeitpunkt t wird ein PUF-Request gestellt, und somit das PUF-Modul stimuliert. Als Folge der Stimulation gibt das Modul den aktuellen PUF-Wert A_t aus.

[0041] Ferner sei E_t der Fehlervektor. Der Vektor E_t stellt den Unterschied dar zwischen dem wahren (allerersten) PUF-Wert A und dem aktuell ausgegebenen PUF-Wert A_t.

[0042] Ein Beispiel für das obige:

$$A = (1\,1\,1\,1\,1\,0\,0\,0\,1\,1);\ A_t = (1\,0\,1\,1\,1\,0\,0\,0\,1\,1)$$

[0043] Dann ist

$$E_t\,(0\,1\,0\,0\,0\,0\,0\,0\,0\,0)$$

[0044] Es gilt also:

$$A = A_t + E_t$$

[0045] Einschub: Bei der klassischen (hier nicht verwendeten) Vorgehensweise zur PUF-Rekonstruktion würde folgendermaßen vorgegangen: Nach einer PUF-Stimulation erhält man A_t. Mithilfe eine ECC-Fehlerkorrekturalgorithmus berechnet man aus A_t den Fehlervektor E_t. Den wahren PUF-Wert A erhält man dann durch $A = A_t + E_t$. Mit dem Symbol beziehungsweise Operator „+“ ist immer das bitweise XOR gemeint.

[0046] Man beachte: Der Fehlervektor E_t beschreibt in Ausführungsbeispielen die Differenz (das Delta) zwischen A und A_t. Daher wird E_t auch als Deltat bezeichnet.

[0047] Es gilt also:

$$Delta_t = E_t$$

[0048] Anders als bei der oben beschriebenen klassischen Vorgehensweise geht man bei der Delta-Correction-Methode gemäß Ausführungsbeispielen folgendermaßen vor. Der zum Zeitpunkt t-1 angefallene Fehlervektor $e_{t-1} = Delta_{t-1}$ wird nicht verworfen, sondern in einem Deltaregister bereitgehalten, also gespeichert. Wenn nun zum Zeitpunkt t der aktuelle PUF-Wert A_t vom PUF-Modul ausgegeben wird, versucht man nicht sofort, A_t mit dem ECC-Fehlerkorrekturalgorithmus zu korrigieren.

[0049] Stattdessen berechnt man

$$B_t = A_t + Delta_{t-1}$$

[0050] Dieser Schritt stellt somit eine Art Vorkorrektur dar. In vielen Fällen werden durch diese Vorkorrektur einige Bitfehler in A_t bereits korrigiert. Folglich stellt das Ergebnis B_t eine bessere Näherung an den gesuchten wahren PUF-Wert A dar als die ursprünglich durch Stimulation erzeugte Bitfolge.

[0051] Nun wird B_t dem ECC-Fehlerkorrekturalgorithmus unterworfen. Mit diesem wird der Fehler et von B_t relativ zu A berechnet. Im allgemeinen wird dieser Fehler et kleiner sein, also weniger Einsen enthalten, als E_t.

[0052] Dann ergibt sich der wahre PUF-Wert A aus B_t und et durch

$$A = B_t + e_t$$

[0053] Zuletzt wird das Delta-Register wie folgt aktualisiert. Der momentane Inhalt des Deltaregisters ist $Delta_{t-1}$. Zu diesem Wert wird der berechnete Fehlervektor et addiert, also einer bitweisen XOR-Operation unterzogen.

[0054] Das Ergebnis:

$$Delta_t = Delta_{t-1} + e_t$$

wird als neuer Wert ins Deltaregister geschrieben.

[0055] Für die nächste PUF-Generierung, bei der A_{t+1} von dem PUF-Modul produziert wird, kann das Verfahren wiederholt werden.

[0056] In einem Beispiel wird das obige veranschaulicht:

$$\text{Sei } A = (1\,1\,1\,1\,1\,0\,0\,0\,1\,1)$$

$$Delta_{t-1} = (1\,0\,1\,0\,0\,0\,0\,0\,0\,1)$$

$$A_t = (0\,1\,0\,0\,1\,0\,0\,0\,1\,1) \quad (\text{Beachte: } A_t \text{ hat drei Fehler})$$

[0057] Man berechnet

$$B_t = (1\,1\,1\,0\,1\,0\,0\,0\,1\,0) \quad (\text{Beachte: } B_t \text{ hat nur noch zwei Fehler})$$

[0058] Nun wird B_t mit dem ECC-Algorithmus fehlerkorrigiert.

[0059] Die ECC gibt den Fehlervektor et aus:

$$e_t\,(0\,0\,0\,1\,0\,0\,0\,0\,0\,1)$$

[0060] Die Addition von e_1 und B_t ergibt den wahren PUF-Wert A:

$$B_t = (1\,1\,1\,0\,1\,0\,0\,0\,1\,0)$$
$$+e_t\,(0\,0\,0\,1\,0\,0\,0\,0\,0\,1)$$
$$- - - - - - - - - - - - - -$$
$$A = (1\,1\,1\,1\,1\,0\,0\,0\,1\,1)$$

[0061] Zuletzt berechnt man Deltat durch Addition von $Delta_{t-1}$ und e_t:

$$Delta_{t-1} = (1\,0\,1\,0\,0\,0\,0\,0\,0\,1)$$
$$+e_t\,(0\,0\,0\,1\,0\,0\,0\,0\,0\,1)$$
$$- - - - - - - - - - - - - - - - - -$$
$$Delta_t\,(1\,0\,1\,1\,0\,0\,0\,0\,0\,0)$$

[0062] Im Allgemeinen wird die Delta-Vorkorrektur eine Erleichterung für den Fehlerkorrekturalgorithmus danach darstellen.

[0063] Es sind aber auch Fälle denkbar, wo die Delta-(Vor)korrektur kontraproduktiv ist: Angenommen, die Zeitpunkte t-1 und t liegen weit auseinander und es würden an diesen Zeitpunkten gänzlich andere Umweltbedingungen herrschen, typischerweise Temperaturen. Dann werden höchstwahrscheinlich auch die Differenzen $Delta_{t-1}$ und Deltat nicht mehr ähnlich sein, sondern sich beträchtlich voneinander unterscheiden. Dann wäre der Vektor B_t eine schlechtere Approximation an den wahren PUF-Wert A als A_t selbst. Es ist in solchen Fällen besser, die Delta-Korrektur zu unterdrücken bzw. wegzulassen und klassisch vorzugehen. Das bedeutet, den Vektor A_t wie bei der oben geschilderten konventionellen Vorgehensweise bei der PUF-Reproduktion direkt in die ECC-Fehlerkorrektur einzuspeisen.

[0064] Daher ist empfehlenswert, den Delta-Korrektur-Mechanismus nicht starr zu implementieren, sondern so, dass er gegebenenfalls auch (automatisch) abschaltbar ist.

[0065] Sinnvoll ist dabei typischerweise folgende Vorgangsweise. Man versucht zuerst die oben beschriebene PUF-Rekonstruktion mit der Delta-Korrektur. Sollte dies misslingen, also die ECC-Fehlerkorrektur anschließend einen Wert ergeben, der in der Folge als falsch erkannt wird, dann versucht man es noch einmal ohne Delta-Korrektur, also auf konventionelle Weise.

[0066] Denkbar ist auch die umgekehrte Reihenfolge, also zuerst die konventionelle Variante des Verfahrens, und anschließend mit der oben beschriebenen Delta-Korrektur.

[0067] Die **Fig. 1** zeigt ein Verfahren **100** zur Rekonstruktion einer PUF A für die Nutzung in einem elektronischen Gerät gemäß Ausführungsbeispielen. Das Verfahren umfasst das Erzeugen einer (potenziell) fehlerbehafteten PUF A_t in einem Block **110**; das Vorkorrigieren der PUF A_t mittels eines gespeicherten Korrekturvektors $Delta_{t-1}$, um eine vorkorrigierte PUF B_t zu erhalten, in einem Block **120**; sowie das Rekonstruieren der PUF A aus der vorkorrigierten PUF B_t mittels eines Fehlerkorrekturalgorithmus in einem Block **130**.

[0068] Das Rekonstruieren der PUF A aus PUF B_t mittels dem Fehlerkorrekturalgorithmus umfasst dabei typischerweise das Anwenden eines ECC-Algorithmus auf die Bitfolge B_t, um als Resultat einen Fehlervektor et zu erhalten. Anschließend wird eine XOR-Verknüpfen von B_t und et durchgeführt. Dadurch erhält man eine Vorstufe des neuen Korrekturvektors Delta. Mittels XOR-Verknüpfung des gespeicherten Korrekturvektors $Delta_{t-1}$ und des Fehlervektors et berechnet man den neuen bzw. aktualisierten Korrekturvektor $Delta_t$. Deltat wird nun als neuer Korrekturvektor in einem Fehlerregister **230** abgespeichert, der bei einer späteren erneuten PUF-Rekonstruktion wiederum als Fehlervektor dient, wie bereits beschrieben. Er übernimmt also die Rolle des früheren $Delta_{t-1}$. In den ECC-Algorithmus werden als Eingabewerte die vorkorrigierte PUF B_t und der Redundanzwert R eingespeist. Letzteres ist detailliert weiter oben beschrieben und soll an dieser Stelle nicht weiter im Detail behandelt werden.

[0069] **Fig. 2** zeigt schematisch ein Ablaufdiagramm eines Verfahrens **200** gemäß Ausführungsbeispielen. In Block **210** wird eine potenziell fehlerbehaftete PUF A_t erzeugt. Diese wird in Block **220** mit dem in Speicher **230** gespeicherten Korrekturvektor $Delta_{t-1}$ XOR-verknüpft. Das Ergebnis ist die vorkorrigierte Bitfolge B_t. Diese wird nun in Schritt **235** einem Fehlerkorrekturalgorithmus unterzogen, typischerweise einem ECC-Algorithmus. Daraus resultiert der Fehlervektor et. Dieser wird in Schritt **240** per XOR-Verknüpfung mit der vorkorrigierten Bitfolge B_t verknüpft, woraus sich die wahre PUF A ergibt. Der Fehlervektor et wird in Block **260** per XOR-Verknüpfung mit dem bereits zuvor benutzten Korrekturvektor $Delta_{t-1}$ per XOR verknüpft, um einen neuen beziehungsweise aktuellen (Vor-)Korrekturvektor Deltat zu erhalten. Dieser wird als neuer Korrekturvektor Deltat in den Speicher **230** geschrieben, Block **270**, um für den nächsten PUF-Rekonstruktionszyklus zur Verfügung zu stehen.

[0070] Die Länge von A_t beträgt dabei typischerweise etwa 64 Bit bis 6000 Bit. Die durchschnittliche Bitfehlerrate von A_t kann dabei von etwa 0,3 % bis etwa 25 % betragen, zum Beispiel 1 %, 3 %, 5 %, 10 %, 15 %, oder 20 %. Die Länge der rekonstruierten wahren PUF A beträgt typischerweise von etwa 64 Bit bis etwa 6000 Bit.

[0071] Typischerweise ist das Verfahren zur PUF-Rekonstruktion gemäß Ausführungsbeispielen auf der Elektronik einer Chipkarte realisiert. Das Erzeugen der fehlerbehafteten PUF A_t geschieht dabei typischerweise durch ein transistorbasiertes Verfahren. Der Fehlervektor Deltat wird üblicherweise auf einem Festwertspeicher gespeichert, zum Beispiel einem EEPROM.

[0072] Wie bereits erwähnt, kann bei ungünstigen Umweltbedingungen, etwa einem großen Temperaturunterschied der Elektronik im Vergleich zur letzten PUF-Rekonstruktion, das Delta-Korrekturverfahren gemäß Ausführungsbeispielen auch die Rekonstruktion negativ beeinflussen, so dass im ungünstigsten Fall gar keine korrekte Rekonstruktion möglich ist. Daher kann in Ausführungsbeispielen standardmäßig getestet werden, ob die PUF A korrekt rekonstruiert wurde. Im Falle einer inkorrekt rekonstruierten PUF A kann dann gegebenenfalls die Rekonstruktion ohne Korrigieren der PUF A_t mittels des gespeicherten Korrekturvektors $Delta_{t-1}$ erneut durchgeführt werden.

[0073] Zu diesem Zweck kann etwa der auf B_t angewendete ECC-Algorithmus mit verschiedenen Stärken vorgesehen sein, so dass im Standardfall (mit Delta-Vorkorrektur) ein schwacher Korrekturalgorithmus zum Einsatz kommt. Versagt dieses Verfahren in Ausnahmefällen wie oben beschrieben, kann auf die Korrektur verzichtet und ein stärkerer Korrekturalgorithmus angewendet werden.

[0074] Die Länge des Korrekturvektors Delta ist in Ausführungsbeispielen identisch zu der Länge des PUF-Wertes A. A kann in der praktischen Anwendung in gleichlange Einzelsegmente zerlegt werden; dementsprechend kann auch Delta in eine identische Mehrzahl von gleichlangen Einzelsegmenten zerlegt werden. Zum Beispiel kann eine 1000 Bit lange PUF in 25 Segmente zu je 40 Bit zerlegt werden, ebenso das gleichlange Delta. In diesem Fall kann ein kleinerer Fehlerkorrekturalgorithmus (der auf kleinere Codewortlängen operiert) angewendet werden, da immer nur die Fehler des betreffenden Einzelsegments korrigiert werden, und nicht alle Fehler der gesamten Bitfolge gleichzeitig.

[0075] Aus der mit dem Verfahren gemäß Ausführungsbeispielen rekonstruierten PUF A wird üblicherweise ein kryptographischer Schlüssel erzeugt. Dieser kann dann zur Verschlüsselung mit einer Blockverschlüsselung oder einer Stromverschlüsselung genutzt werden, typischerweise AES oder Triple-DES.

[0076] **Fig. 3** zeigt schematisch eine Vorrichtung **300** gemäß Ausführungsbeispielen, die zur Ausführung eines Verfahrens gemäß Ausführungsbeispielen ausgelegt ist. Sie kann zum Beispiel eine Chipkarte oder ein Kryptographiemodul sein, das in einem Endgerät eingebettet ist, zum Beispiel einem mobilen Endgerät wie einem Mobiltelefon oder einem RFID-Gerät.

[0077] Die Vorrichtung umfasst ein Element **310** zur Erzeugung einer potenziell fehlerbehafteten PUF A_t, einen Speicher **320** für einen Korrekturvektor $Delta_{t-1}$, sowie eine Recheneinheit **340** zur Ausführung eines Fehlerkorrekturalgorithmus. Eine weitere Recheneinheit **350** dient zur Durchführung der beschriebenen XOR-Operationen und aller weiteren, teilweise hier nicht detailliert beschriebenen Operationen, die dem Fachmann bekannt sind. Die beiden Recheneinheiten bzw. alle Bestandteile der Vorrichtung **300** können in Ausführungsbeispielen integriert ausgeführt sein, etwa auf einem integrierten Schaltkreis **360**. Dieser kann auch weitere Bestandteile umfassen, z.B. die Logik für die Rekonstruktion des kryptographischen Schlüssels und andere typische Bestandteile einer Chipkarte bzw. eines Kryptografiemoduls, wie dem Fachmann bekannt.

[0078] Dem Fachmann ist ohne weiteres verständlich, dass die hier beschriebenen Verfahren und Vorrichtungen auch für andere als die hier beschriebenen Zwecke einsetzbar sind, und dass sie in unterschiedlichsten elektronischen Geräten einsetzbar sind, bei denen eine PUF rekonstruiert werden soll. Zudem lassen sich die hierin angegebenen Schlüssellängen und Längen der Bitfolgen mit zum Standardwissen des Fachmanns gehörenden Methoden für andere als die hierin konkret beschriebenen Fälle modifizieren.

Patentansprüche

1. Verfahren (100, 200) zur Rekonstruktion einer PUF A für die Nutzung in einem elektronischen Gerät, umfassend:
- Erzeugen (110) einer potenziell fehlerbehafteten PUF A_t;

- Vorkorrigieren (120, 220) der PUF A_t mittels eines gespeicherten Korrekturvektors $Delta_{t-1}$, um eine vorkorrigierte PUF B_t zu erhalten;
- Rekonstruieren (130, 235) der PUF A aus der vorkorrigierten PUF B_t mittels eines Fehlerkorrekturalgorithmus, umfassend:
- Anwenden (235) des Fehlerkorrekturalgorithmus auf B_t, um einen Fehlervektor e_t zu erhalten;
- XOR-Verknüpfen (240) von B_t und e_t,

wobei ein neuer Korrekturvektor Deltat durch XOR-Verknüpfung (260) des gespeicherten Korrekturvektors $Delta_{t-1}$ und des Fehlervektors e_t berechnet wird, und wobei Deltat als neuer Korrekturvektor in einem Fehlerregister abgespeichert (270) wird, der bei einer späteren erneuten PUF-Rekonstruktion als Fehlervektor $Delta_{t-1}$ dient.

2. Verfahren nach Anspruch 1, wobei die Länge von A_t 64 Bit bis 6000 Bit beträgt.

3. Verfahren nach einem der vorhergehenden Ansprüche, wobei die durchschnittliche BitFehlerrate von A_t 0,3 % bis 25 % beträgt.

4. Verfahren nach einem der vorhergehenden Ansprüche, wobei die Länge von A 64 Bit bis 6000 Bit beträgt.

5. Verfahren nach einem der vorhergehenden Ansprüche, wobei das Verfahren auf einer Chipkarte (300) oder einem RFID-Gerät ausgeführt wird.

6. Verfahren nach einem der vorhergehenden Ansprüche, wobei das Erzeugen (110, 210) der fehlerbehafteten PUF A_t durch ein transistorbasiertes Verfahren geschieht.

7. Verfahren nach einem der vorhergehenden Ansprüche, wobei der Fehlervektor E auf einem Festwertspeicher (320) gespeichert ist, vorzugsweise einem EEPROM.

8. Verfahren nach einem der vorhergehenden Ansprüche, wobei getestet wird, ob die PUF A korrekt rekonstruiert wurde.

9. Verfahren nach Anspruch 8, wobei im Falle einer inkorrekt rekonstruierten PUF A die Rekonstruktion ohne Korrigieren der PUF A_t mittels eines gespeicherten Korrekturvektors $Delta_{t-1}$ erneut durchgeführt wird.

10. Verfahren nach einem der vorhergehenden Ansprüche, wobei der Fehlerkorrekturalgorithmus ein ECC-Algorithmus ist.

11. Verfahren nach einem der vorhergehenden Ansprüche, wobei die Länge des Korrekturvektors Delta von 64 Bit bis 6000 Bits beträgt.

12. Verfahren nach einem der vorhergehenden Ansprüche, weiter umfassend:
- Erstellen eines kryptographischen Schlüssels aus der PUF A.

13. Verfahren nach Anspruch 12, wobei der kryptographische Schlüssel zur Verschlüsselung mit einer Blockverschlüsselung oder einer Stromverschlüsselung genutzt wird, vorzugsweise AES oder Triple-DES.

14. Verfahren nach einem der vorhergehenden Ansprüche, wobei die PUF A_t und der Korrekturvektor Delta in gleichlangen Segmenten segmentweise verarbeitet werden.

15. Eine Vorrichtung (300), umfassend:
- ein Element (310) zur Erzeugung (110, 210) einer potenziell fehlerbehafteten PUF A_t;
- einen Speicher (320) für einen Korrekturvektor $Delta_{t-1}$;
- eine Recheneinheit (340) zur Ausführung (130, 230) eines Fehlerkorrekturalgorithmus; wobei die Vorrichtung zur Ausführung eines Verfahrens gemäß einem der Ansprüche 1 bis 14 ausgelegt ist.

16. Eine Vorrichtung nach Anspruch 15, wobei die Vorrichtung eine Chipkarte oder ein RFID-Gerät ist.

Es folgen 3 Seiten Zeichnungen

Anhängende Zeichnungen

Fig. 1

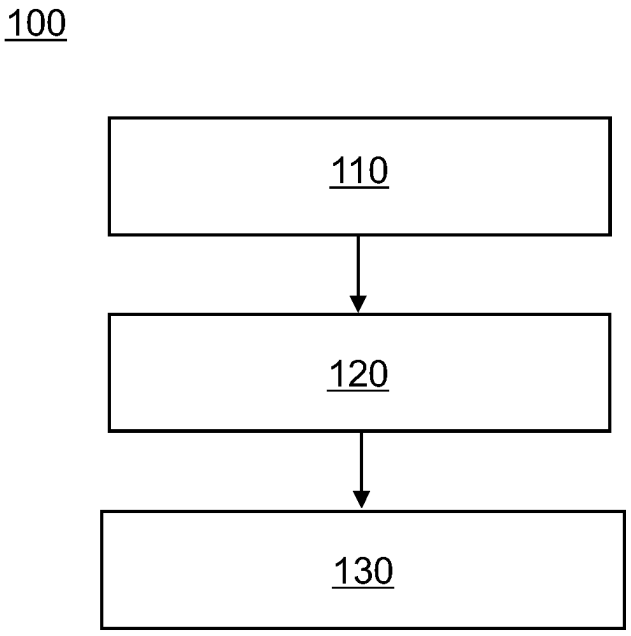


Fig. 2

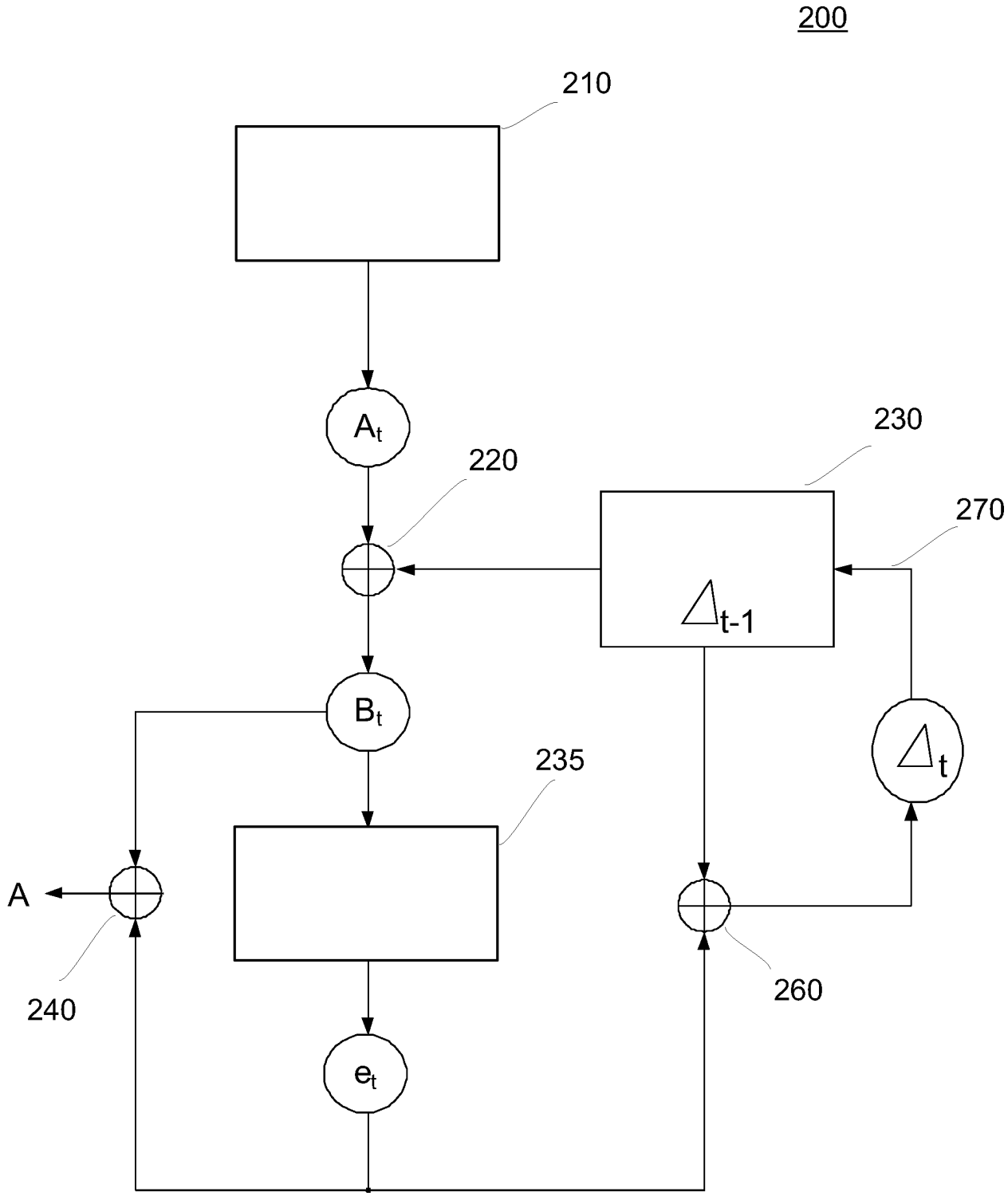


Fig. 3